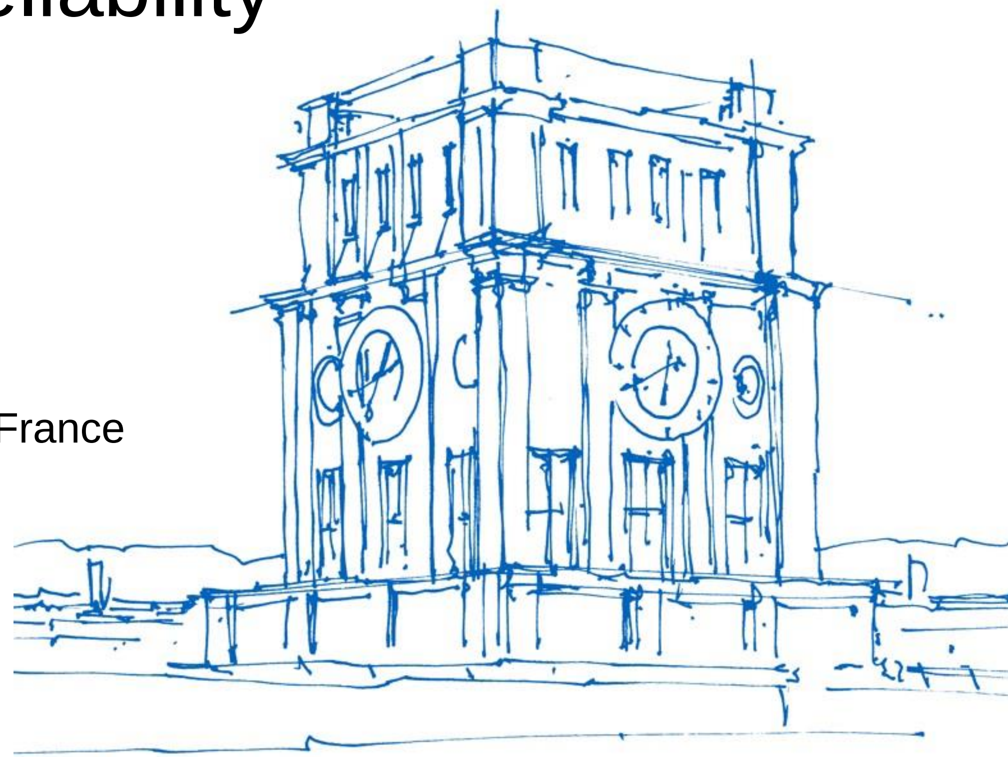
Chair of Communication Networks
School of Computation, Information and Technology
Technical University of Munich

# Impact of Software Availability on System Reliability

Carmen Mas-Machuca
Shakthivelu Janardhanan
Yagiz Özkan

September 19-21, 2022, Compiegne, France

***Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of the project "6G Future Lab Bavaria"***
***German Research Fundation (DFG) under grant numbers MA 6529/4-1 and KE 1863/10-1.***

# Ubiquity and magnitude of software failures

- Software bugs contribute more than 35% of critical network outages [Google2016]
- According to Gartner, the average cost of IT downtime is $5,600 per minute. Amazon may lose millions$ in an hour [Forbes Technology Council, April 2021]

**02.09.2021 0**

AWS Direct

Event in the T

NORTHEAST-

https://aws.an

de/messag

**14.07.22 7:49AM(CET).**

Twitter outage brings th

https://bgr.com/tech/tw

around-the-world-right-

**08.05.22 23:45 to 09.05.2022 1:4**

Google Infrastructure Config

failing

**Incident affecting Google Cloud**

**Directory, Cloud CDN, Cloud Loa**

**Approval, Google App Engine, A**

https://status.cloud.google.com

**10.12.21**

the Canadian Centre for Cyber Security (CCCS) issued a security advisory regarding a **critical vulnerability** → Apache Log4j, a widely used open-source tool for logging and recording activity→ would-be attackers to run malicious code on a remote device.→ Quebec shut down almost 4,000 websites.

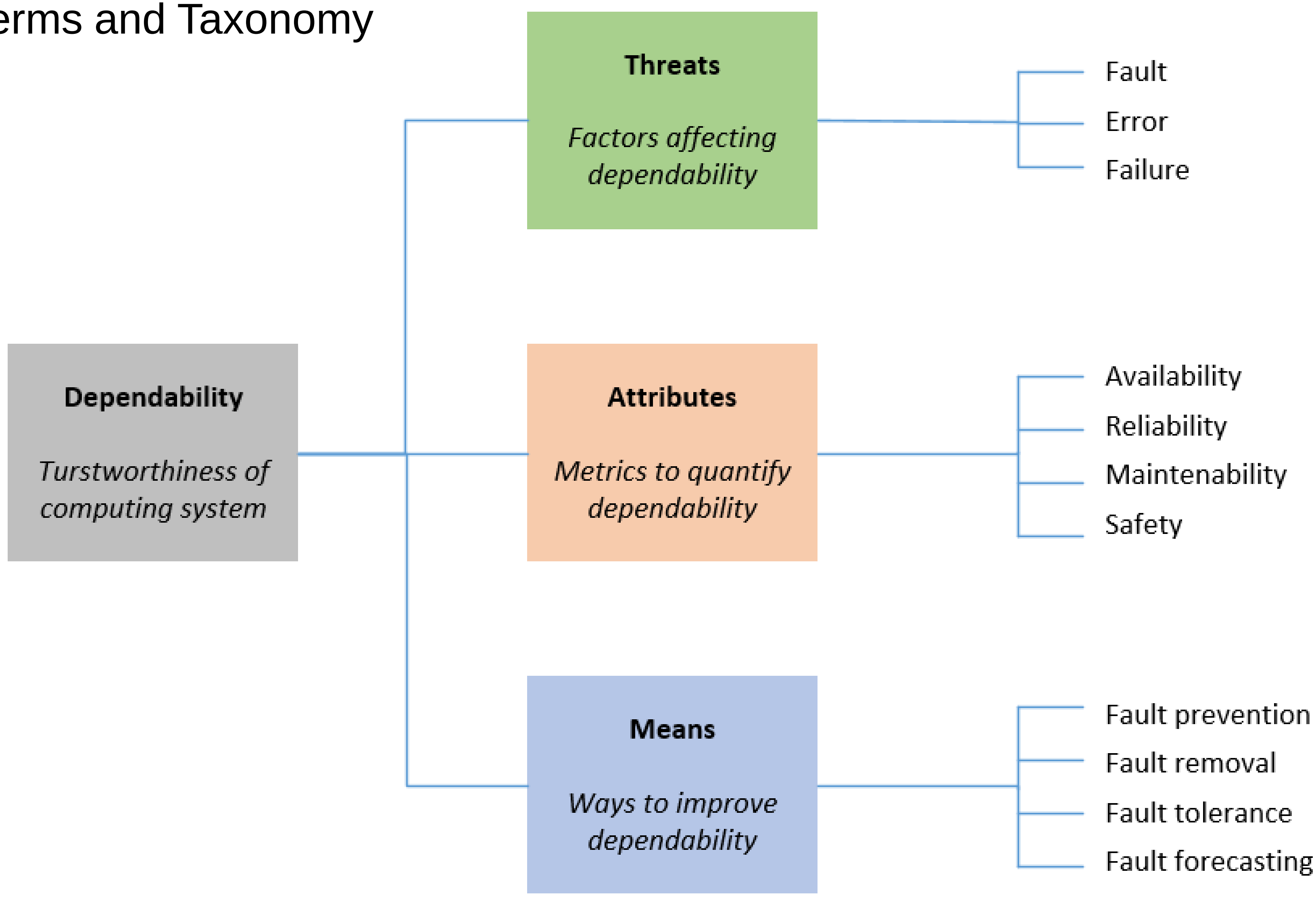
https://carleton.ca/polisci/?p=33162

PST: Pacific Standard Time
CET: Central European Time

# Outline

- Terms and Taxonomy
- Software Dependability Problem
- Addressed questions:
  - How reliable is a new software release?
  - How reliable is a component?
  - How reliable is a system?
- Conclusions

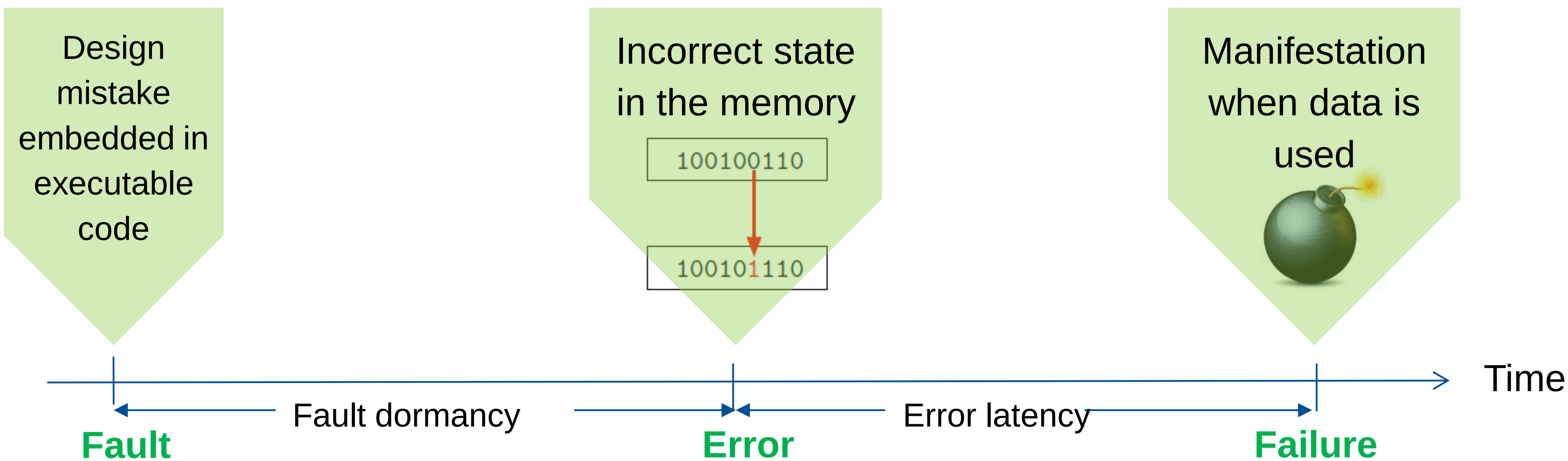
# Terms and Taxonomy




Source: IFIP WG10.4 Dependable Computing and Fault Tolerance https://www.dependability.org/wg10.4/

# Terms and Taxonomy


- **Fault**: Adjudged or hypothesized cause of an error.
- **Error**: Part of a system state which is liable to lead to failure.
- **Failure**: Deviation of the delivered service according to its specification.




- Active: it produces an error
- Dormant: it has not produced an error

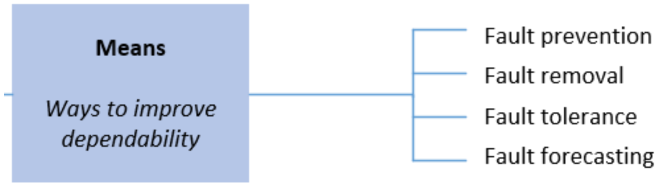
- Detected: it has manifestated as failure
- Latent: it has not been detected

# Terms and Taxonomy


- **Availability**: The ability of an item to perform its required function, under environmental and operational conditions at a stated instant of time.
- **Reliability**: The ability of an item to perform its required function, under environmental and operational conditions, for a stated period of time.
- **Maintenability**: the probability of performing a successful repair and maintenance action within a given time.
- **Safety**: Ability of an item to provide its required function without the occurrence of catastrophic consequences on the user(s) and the environment.

Source: ISO 8402 and British Standard BS 4778

# Terms and Taxonomy




- **Fault prevention** is attained by quality control techniques employed during the design and manufacturing of hardware and software.
- **Fault removal** is performed both during the development phase (verification, diagnosis, and correction), and during the operational life of a system (either corrective or preventive maintenance).
- **Fault tolerance** is intended to preserve the delivery of correct service in the presence of active faults.
- **Fault forecasting** is conducted by performing an evaluation of the system behaviour with respect to fault occurrence or activation: either qualitative (identify, classify, rank the failure modes), or quantitative (probabilities to which some of the attributes are satisfied).

Source: “Fundamental Concepts of Dependability” A. Avizienis et al.

# Terms and Taxonomy: Software faults


- Software fault = bug

- Types of software faults:

| | Bohrbugs (deterministic) |
|---|---|
| Description | „solid“ logical faults |
| Fault handling strategies | Remove |
| Example | *Path Computation Element (PCE) able to create tunnel with negative bandwidth* |

# Terms and Taxonomy: Software faults


- Software fault = bug

- Types of software faults:

| | Bohrbugs (deterministic) | Mandelbugs (non-deterministic) |
|---|---|---|
| Description | „solid“ logical faults | „relative“ logical faults |
| Fault handling strategies | Remove | Retry, replicate |
| Example | *Path Computation Element (PCE) able to create tunnel with negative bandwidth* | *Distributed database locking in ONOS* |

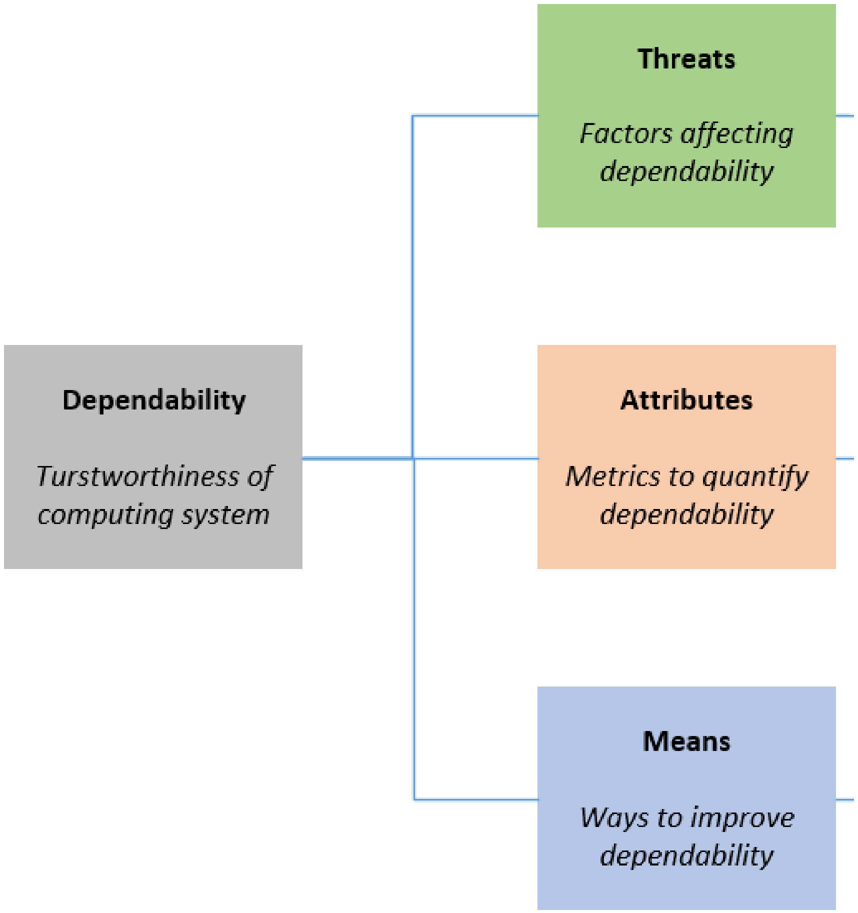
# Terms and Taxonomy: Software faults


- Software fault = bug

- Types of software faults:

| | Bohrbugs (deterministic) | Mandelbugs (non-deterministic) | Ageing-related bugs |
|---|---|---|---|
| Description | „solid“ logical faults | „relative“ logical faults | Degradation with time |
| Fault handling strategies | Remove | Retry, replicate | Rejuvenate |
| Example | *Path Computation Element (PCE) able to create tunnel with negative bandwidth* | *Distributed database locking in ONOS* | *Flows still reported in oper data store after they have been deleted from both config and network* |

# Limitations of the State of the Art




- **Threat** analysis focus on independent component failures
  - Focused on hardware failures
  - Software related failures *neglected* or *oversimplified (e.g., as single failure mode)*
- **Attributes,** e.g.,
  - *reliability*, does not precisely describe software behaviour
    - *Reliability growth* due to *maturity*
    - *Reliability degradation* due to *ageing*
- **Means** focus on structural protection
  - Fault *prevention*, *removal* and *forecasting* have been overlooked

Source: IFIP WG10.4 Dependable Computing and Fault Tolerance https://www.dependability.org/wg10.4/

# Software Dependability Problem

- Softwarized components/systems/networks
- Open source code

# Software Dependability Problem

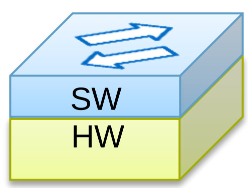
- Softwarized components/systems/networks
- Open source code

**Target**: Realistic and practical dependability analysis

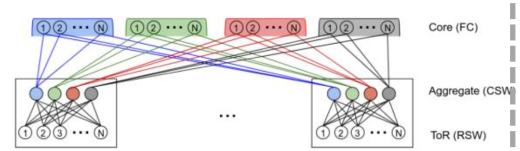
**Specific problems:**

How reliable is a new release?

How reliable is a component?




How reliable is a system?



# Software Dependability Problem

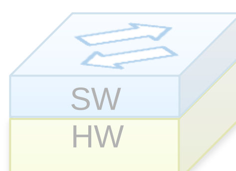
- Softwarized components/systems/networks
- Open source code

**Target**: Realistic and practical dependability analysis

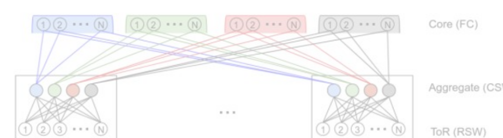
**Specific problems:**

**How reliable is a new release?**

How reliable is a component?




How reliable is a system?

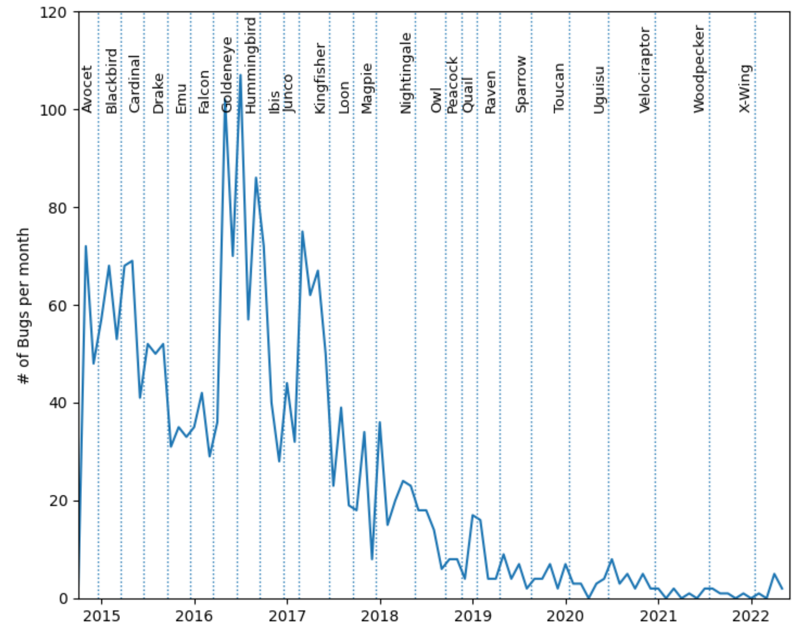


# How reliable is a new release?

**Data Collection**

Bug trackers/
Repositories (e.g., Jira)

Detected bugs
Resolved bugs

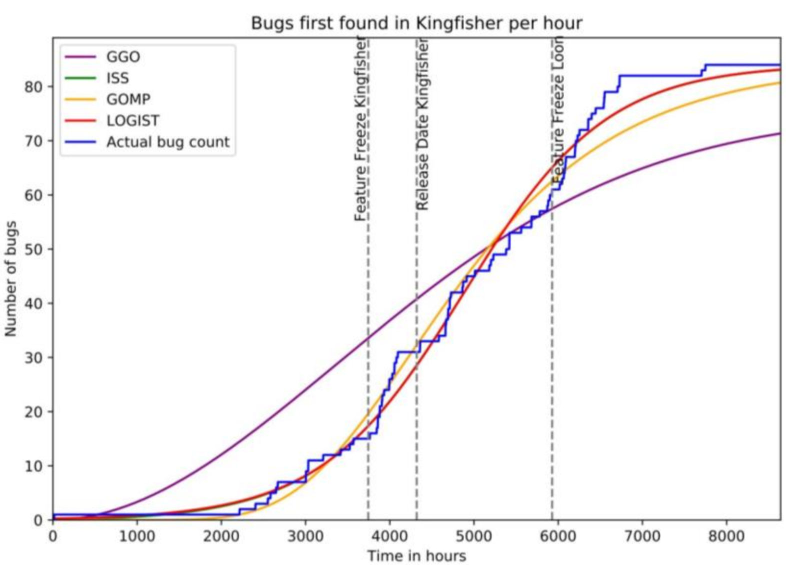



| Issue | ONOS-8153 | ONOS-6401 |
|---|---|---|
| Status | In progress | Closed |
| Priority | Major | Crititcal |
| Affected Versions | 2.3.0 | 1.9.2, 1.10.0, 1.8.5, 1.8.6 |
| First Affected Version | 2.3.0 | 1.8.5 |
| Resolution | Unresolved | Fixed |
| Create Date | 2022-04-08 07:06:16 | 2017-05-00 09:29:49 |
| Create Date from Start | 2916 days | 1114 days |
| Resolved Date | None | 2017-05-02 21:29:49 |
| Time to Solve (in hours) | None | 313.97 |
| Time to Solve (in days) | None | 13 |
| Month Number from Project Start | 96 | 37 |
| Week Number from Project Start | 416 | 159 |

ONOS bugs examples

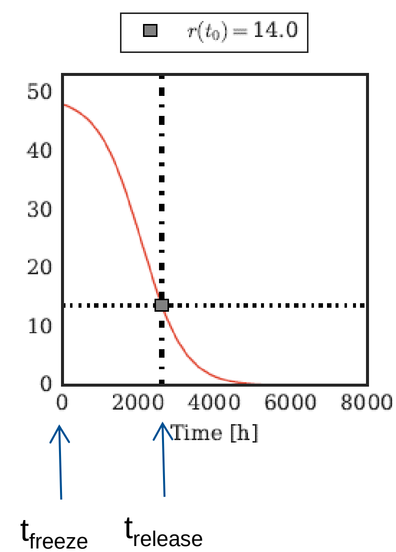
# How reliable is a new release?

| Data Collection | Bug History Analysis |
| --- | --- |
| Bug trackers/ Repositories (e.g., Jira) | Best Models (e.g., SRGM) |
| Detected bugs<br>Resolved bugs | Expected time between detected/resolved bugs<br>Residual bug content |




*Residual bug content*

$$r(t) = E[a - N(t)] = a - m(t)$$




*SRGM: Software Reliability Growth Models*

# How reliable is a new release?

Data Collection

Bug trackers/ Repositories (e.g., Jira)

Bug History Analysis

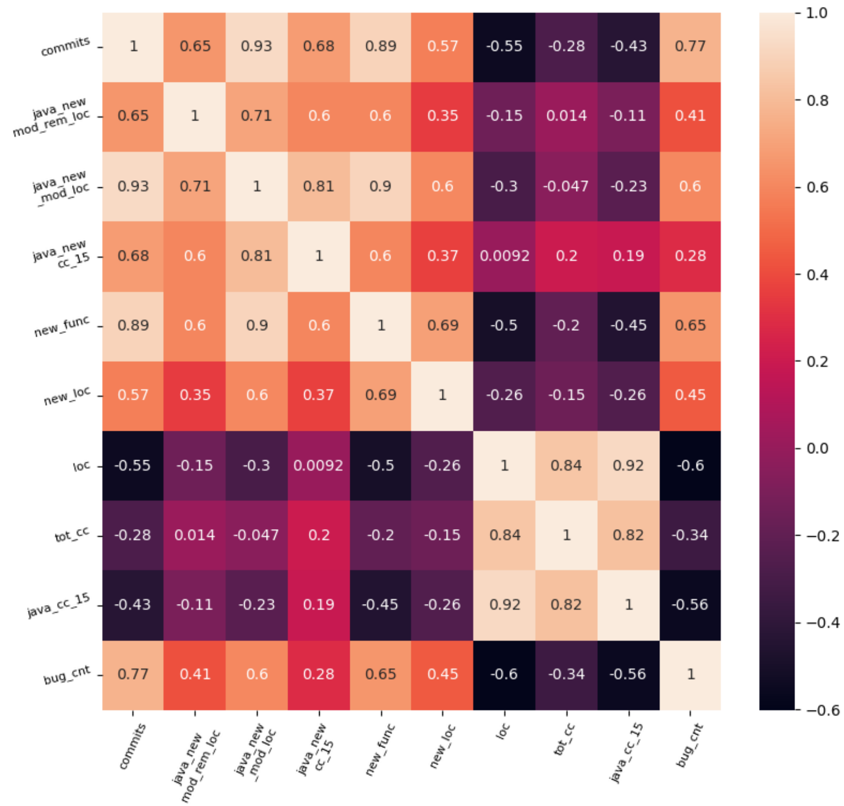
Best Models (e.g., SRGM)

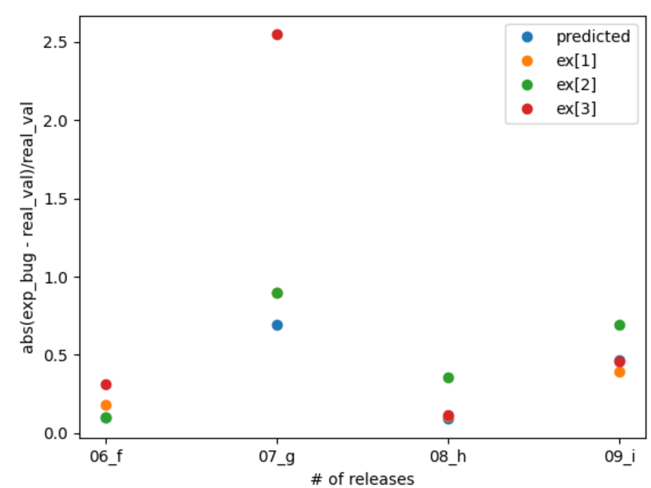
between
ved bugs
content

Estimation accuracy improvement

+Code Metrics

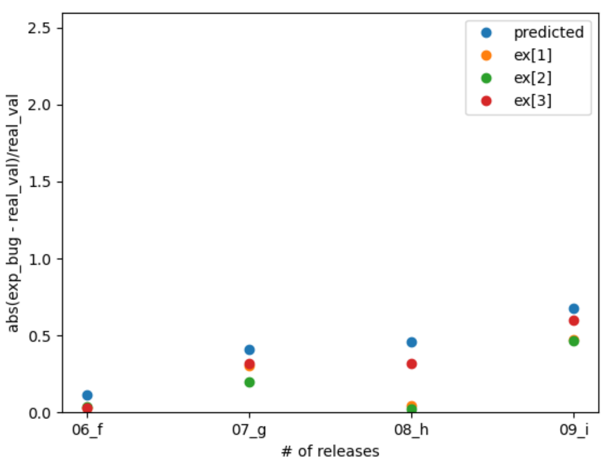
Increase estimation accuracy

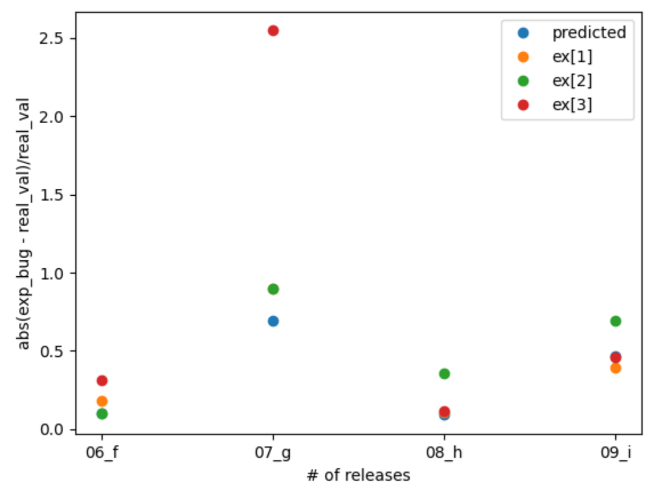


# How reliable is a new release?

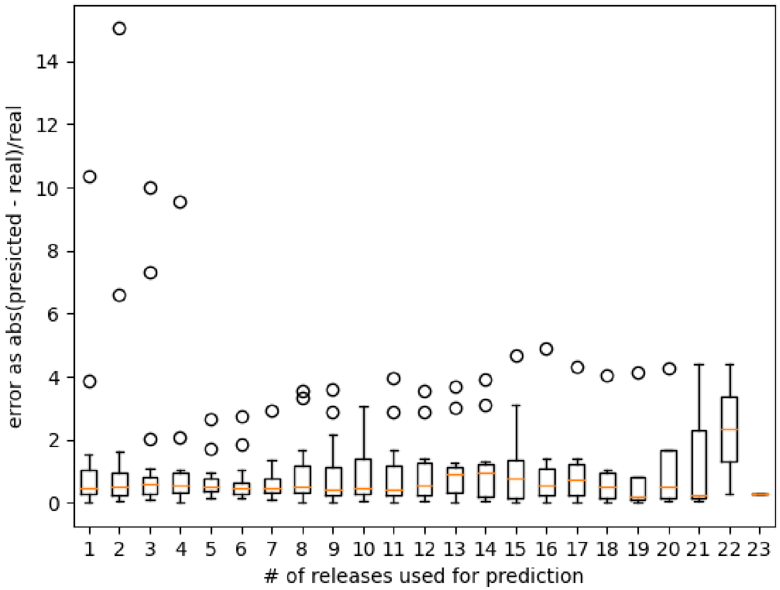
| Data Collection | Bug History Analysis | Estimation accuracy improvement |
|---|---|---|
| Bug trackers/ Repositories (e.g., Jira) | Best Models (e.g., SRGM) | +Code Metrics<br>+Metrics from other codes |
| Detected bugs<br>Resolved bugs | Expected time between detected/resolved bugs<br>Residual bug content | Increase estimation accuracy |



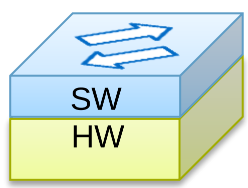
# How reliable is a new release?

| Data Collection | Bug History Analysis | Estimation accuracy improvement |
|---|---|---|
| Bug trackers/ Repositories (e.g., Jira) | Best Models (e.g., SRGM) | +Code Metrics<br>+Metrics from other codes<br>+Metrics from prev. rel. |
| Detected bugs<br>Resolved bugs | Expected time between detected/resolved bugs<br>Residual bug content | Increase estimation accuracy |

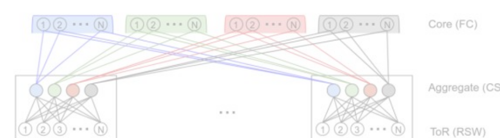


How reliable is a new release?


How reliable is a component?




How reliable is a system?

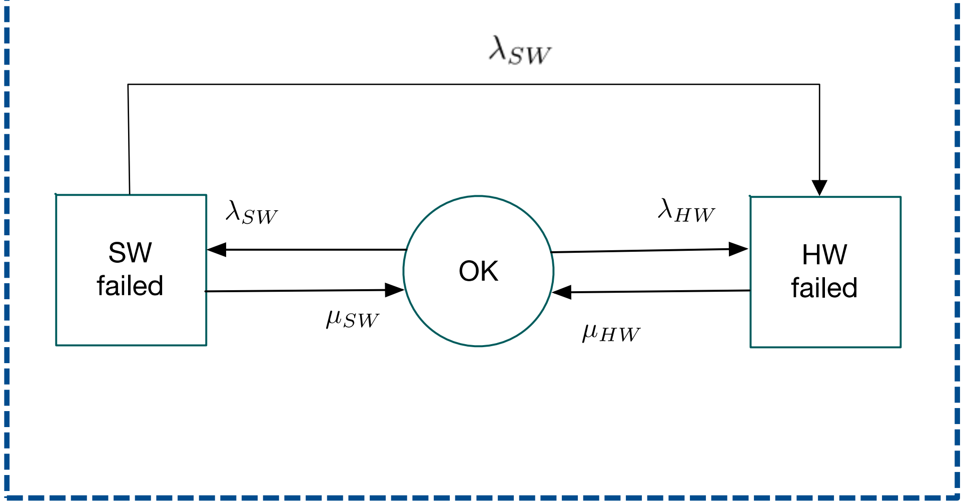


# Component reliability considering software dependability

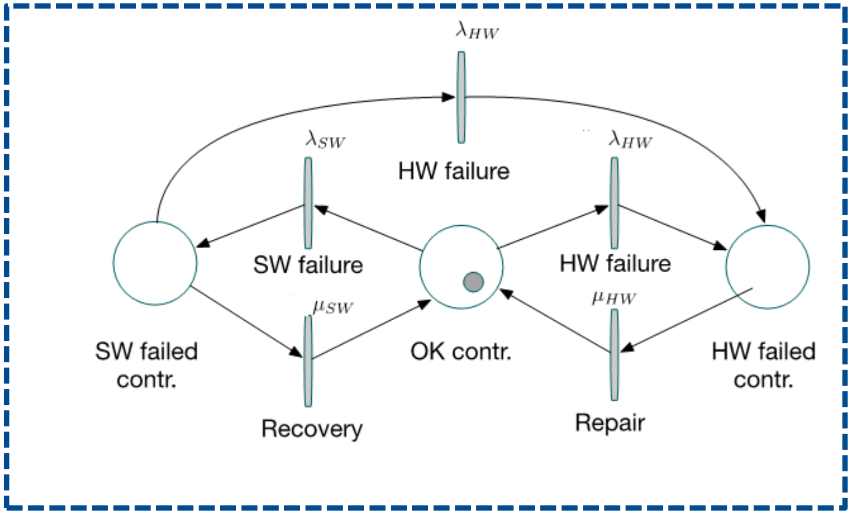
- Hardware & Software
- Software: Propiertary & open-source

Models

Homogeneous Markov Chains

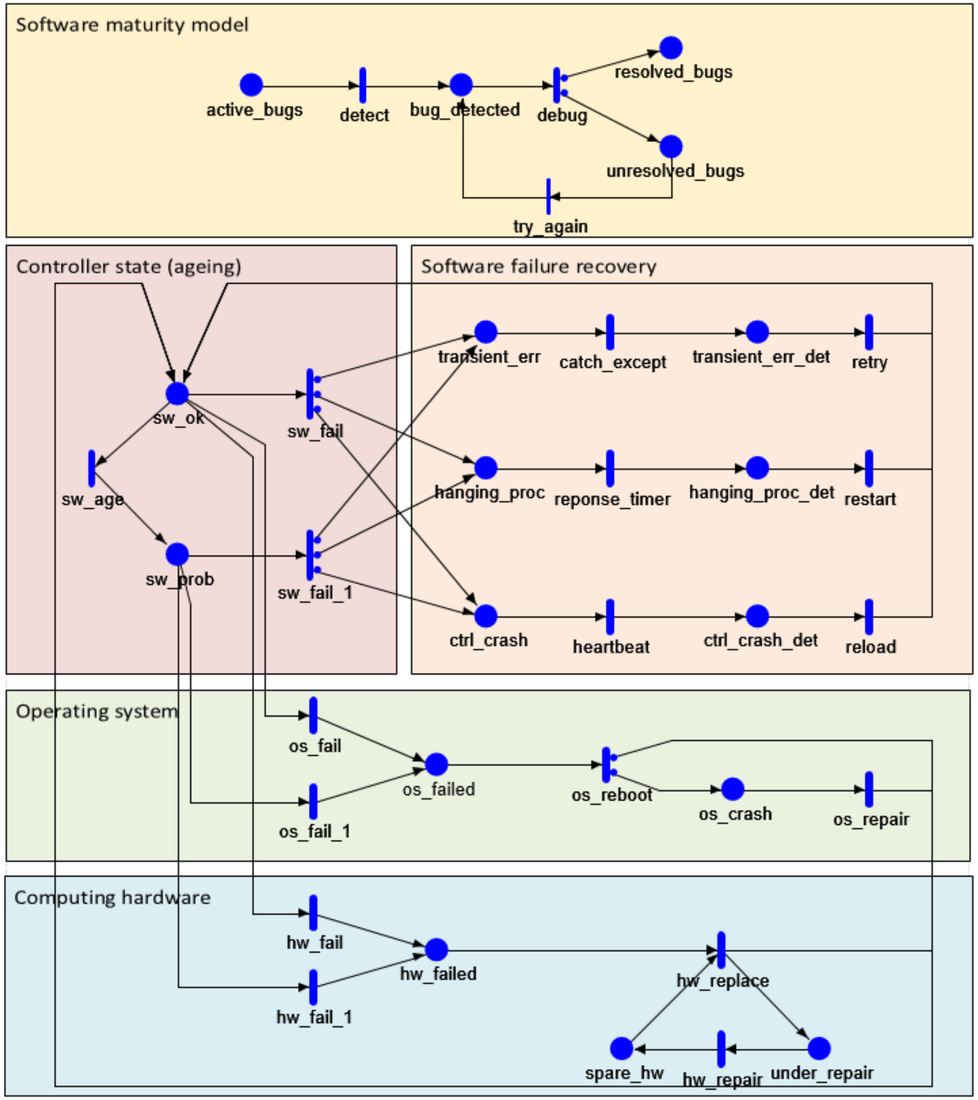



Stochastic Petri Nets/ Stochastic Activity Networks (SANs)



# Component reliability considering software dependability

Example: SDN controller




**1. Software reliability growth**

long term variations of
software reliability

**2. Software ageing**

short term variations of
software reliability

**3. Nature of failures**

**manifestation**
transientfailure
hang and freeze
crash

**recovery**
retry - restart - reload

**4. Operating system**

**5. General purpose Hardware**

# Component reliability considering software dependability

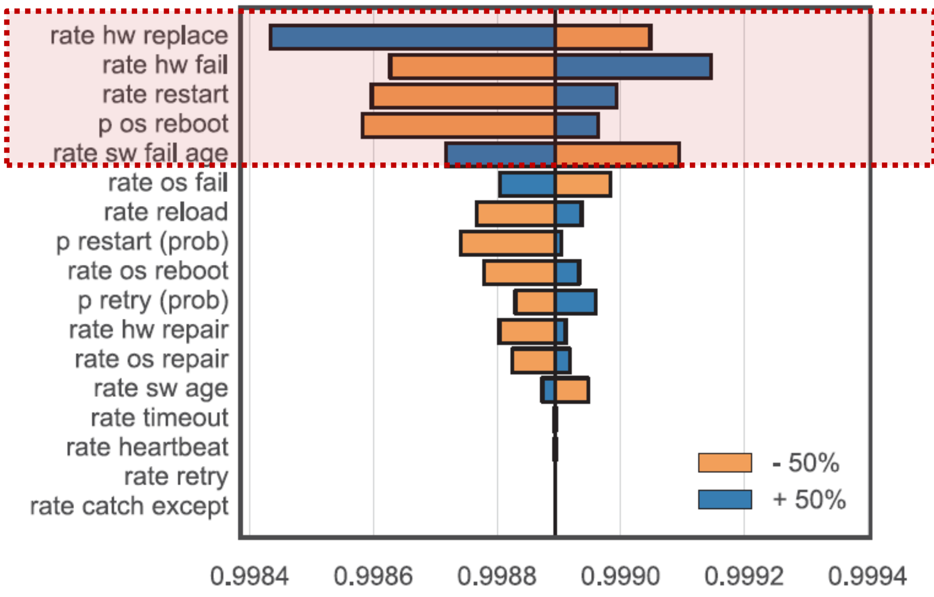
Example: SDN controller→ *SSA analysis*

- At least two controllers are needed to achieve "3-nines" availability

| Component | Controller | SW | OS | HW |
|---|---|---|---|---|
| Availability | 0.99889 | 0.99956 | 0.99981 | 0.99951 |

[Ros14] assumed much higher availability of SDN controller A > 0.999975

- Identification of the most critical parameters (local sensitivity analysis)




**Critical parameters**

a) External failure rates

(well studied and documented)

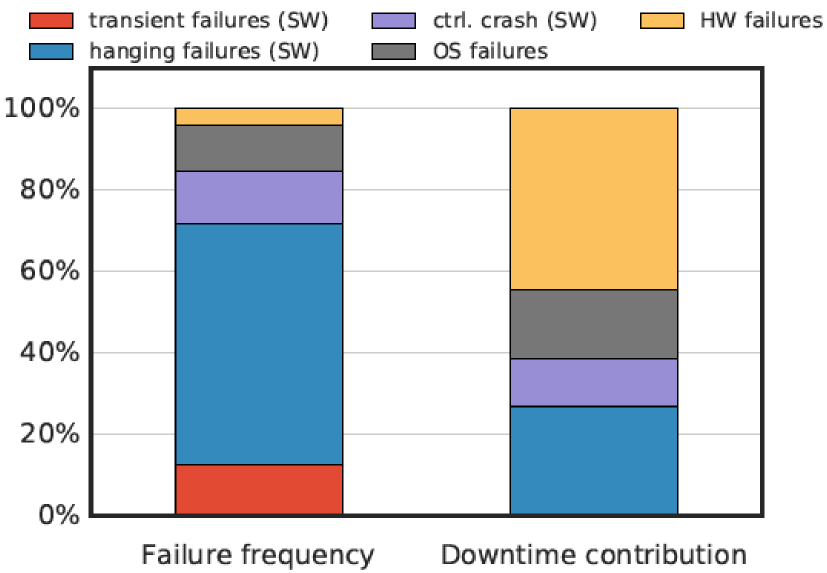
b) <u>Software ageing rate</u>

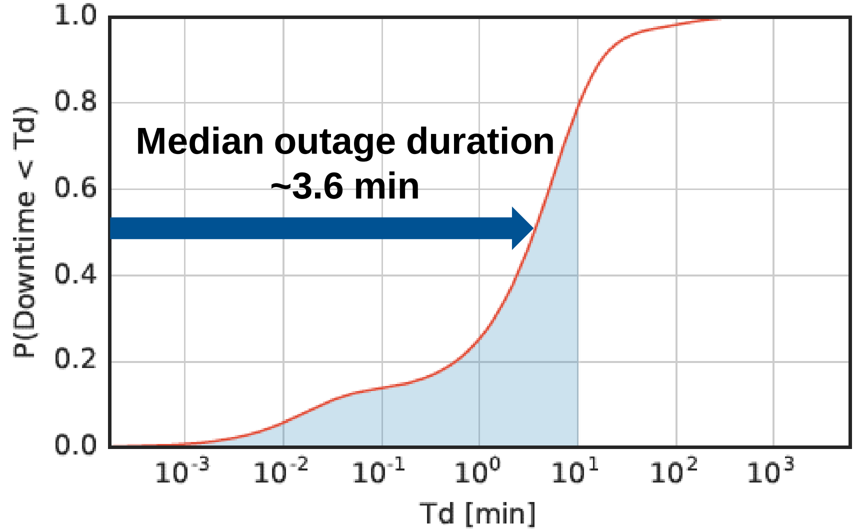
(uncertain, load dependant)

# Component reliability considering software dependability

Example: SDN controller→ *Failure frequency and downtime distribution*

Around 50 failures per year with total duration of 9.68 hours per year are expected.



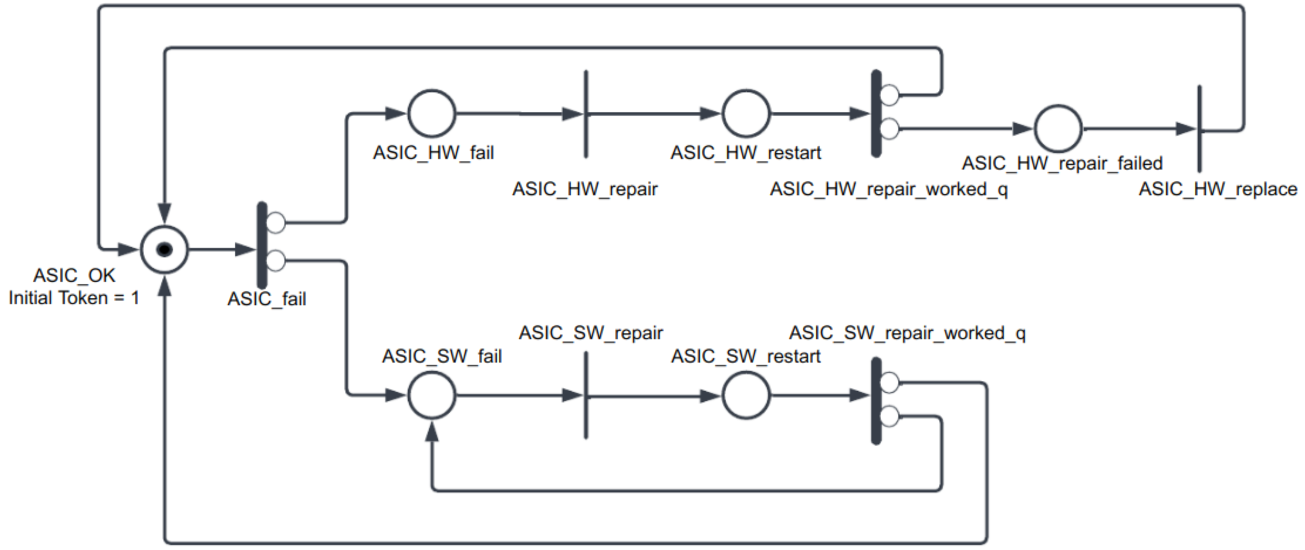




- Software failures lead to more frequent, but shorter, outages
  - Software failures account for 84% of all failures, but contribute to only 38% of downtime
  - Hardware failures represent less then 4% of all failures but contribute to 44% of downtime
  - 80% of the failures resulted in outages shorter than 10 min; median being 3.6 min

# Component reliability considering software dependability

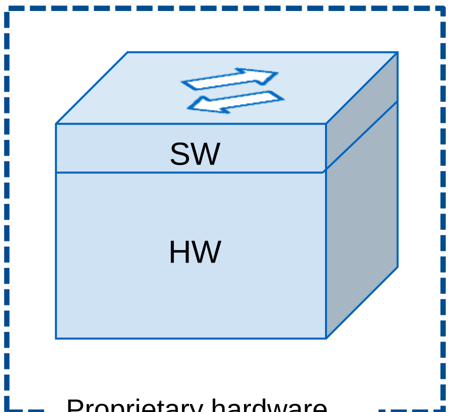
Example: Switch

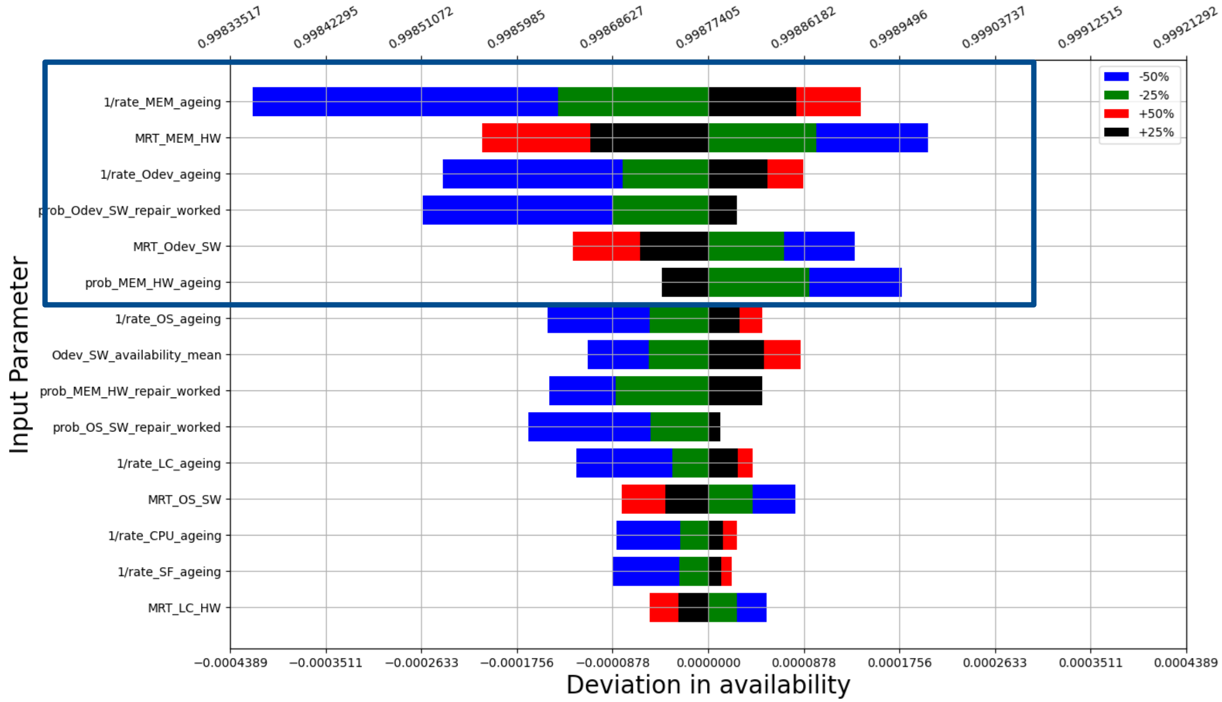
- Several components: ASIC, Memory, CPU, Line Cards, Switch fabric, ..
- Each component:
  - Regular HW and SW failures
  - Ageing for HW and SW



# Component reliability considering software dependability

Example: Switch

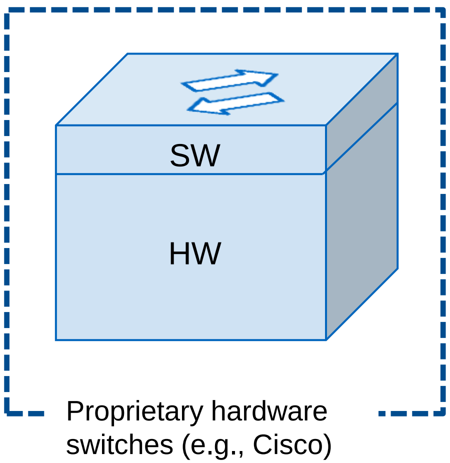


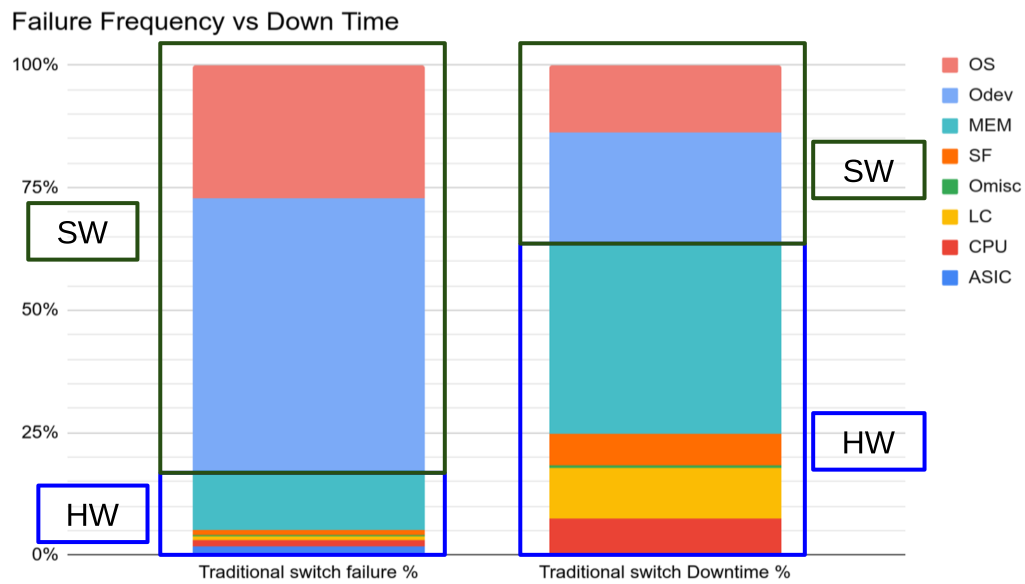




- Most critical parameters:
  - Memory Ageing and HW_reparation times
  - Other SW Dev ageing and successful repair

# Component reliability considering software dependability

## Example: Switch

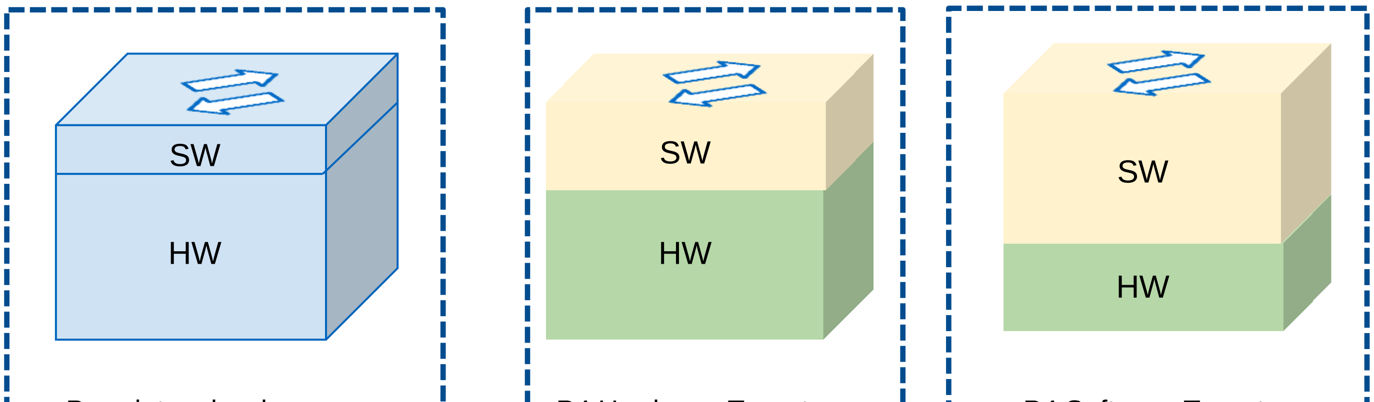



- Software failures lead to more frequent, but shorter outages
  - Software failures account for 80% of all failures, but contribute to only 35% of downtime
  - Hardware failures represent less then 20% of all failures but contribute to 65% of downtime
- Swich availability→ 0,9988 ➡ MDT~10,1 hours/year

# Component reliability considering software dependability

## Example: Switch



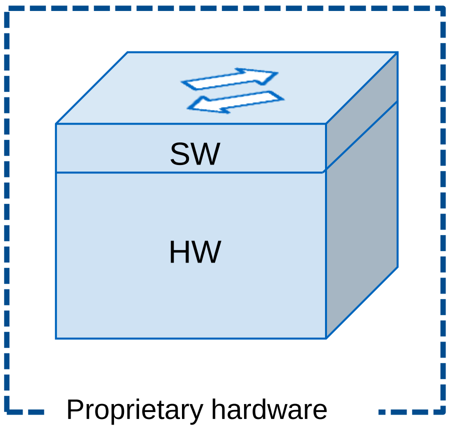

Proprietary hardware switches (e.g., Cisco)

P4 Hardware Target switch (e.g., Intel Tofino)

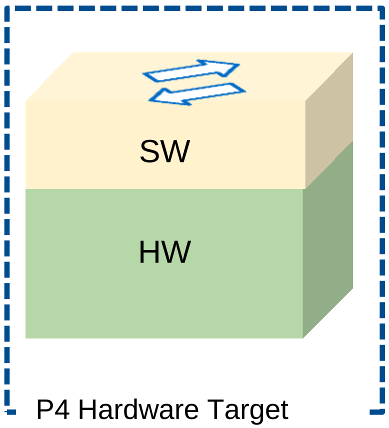
P4 Software Target switch (e.g., t4p4s)

# Component reliability considering software dependability

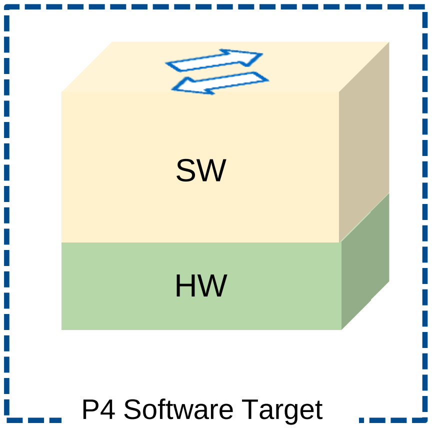
## Example: Switch




Proprietary hardware switches (e.g., Cisco)

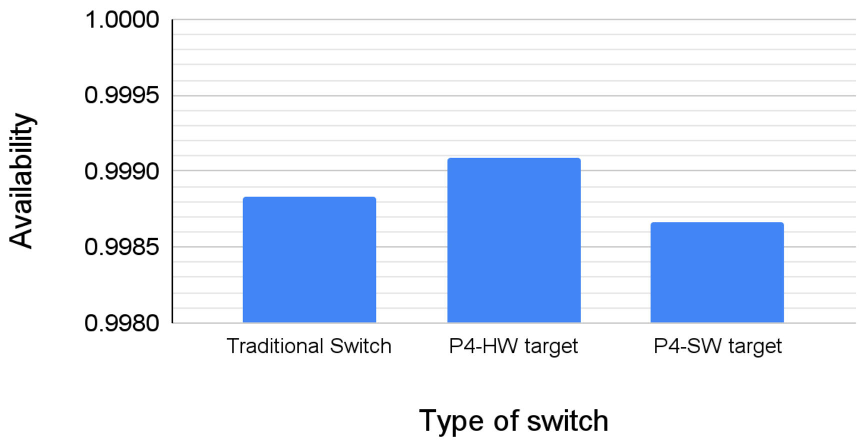



P4 Hardware Target switch (e.g., Intel Tofino)

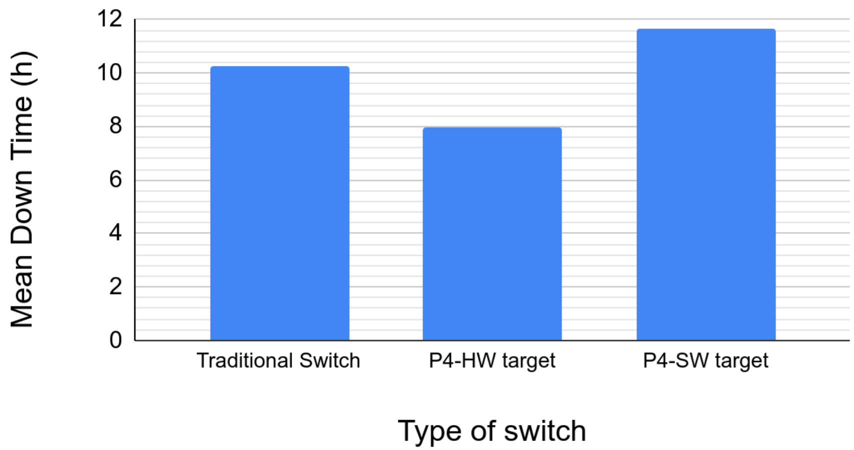



P4 Software Target switch (e.g., t4p4s)

### Switch Availability comparison

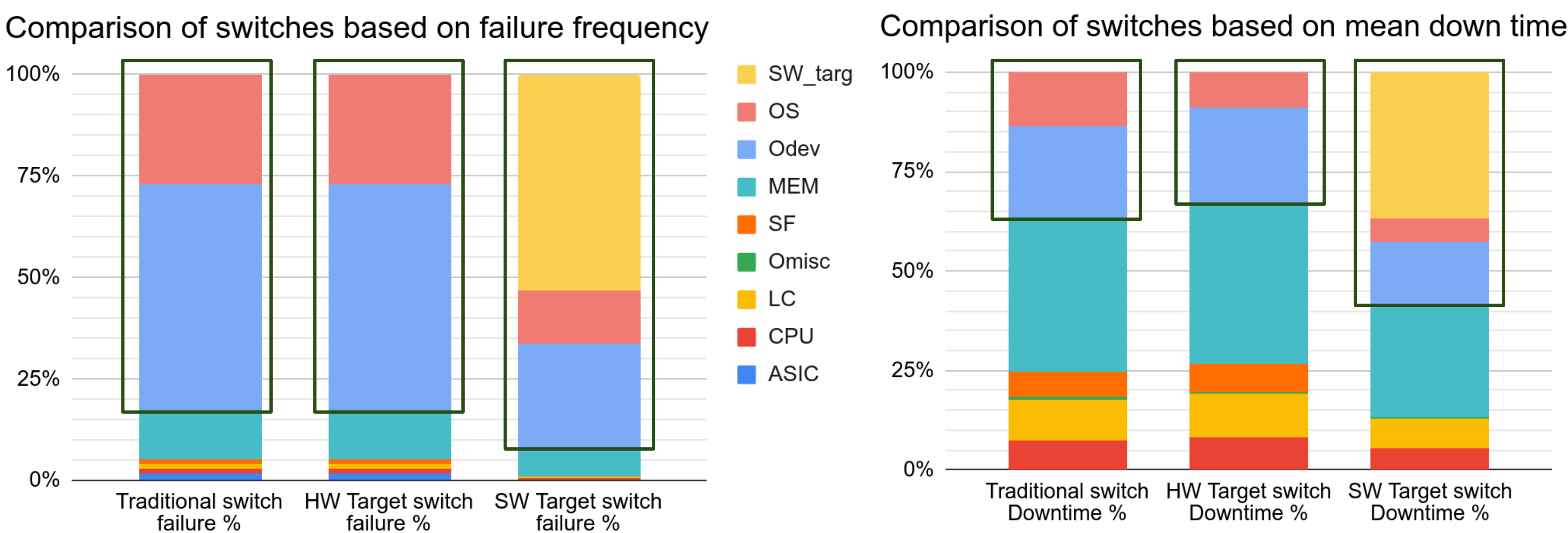



### Switch Mean Down Time comparison



# Component reliability considering software dependability

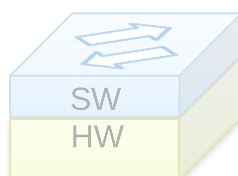
Example: Switch



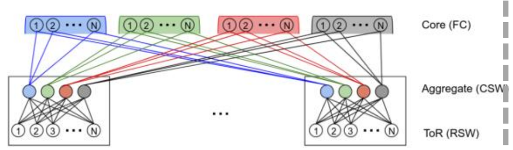

- P4 software target has higher software failure frequency (92%) than other switches (82%)
- Software failures are faster to repair→ P4 software target switch more MDT due to software failures→ SW_targ is the most critical component
- HW Target switch has faster SW restoration time thanks to their modular SW.

How reliable is a new release?

How reliable is a component?

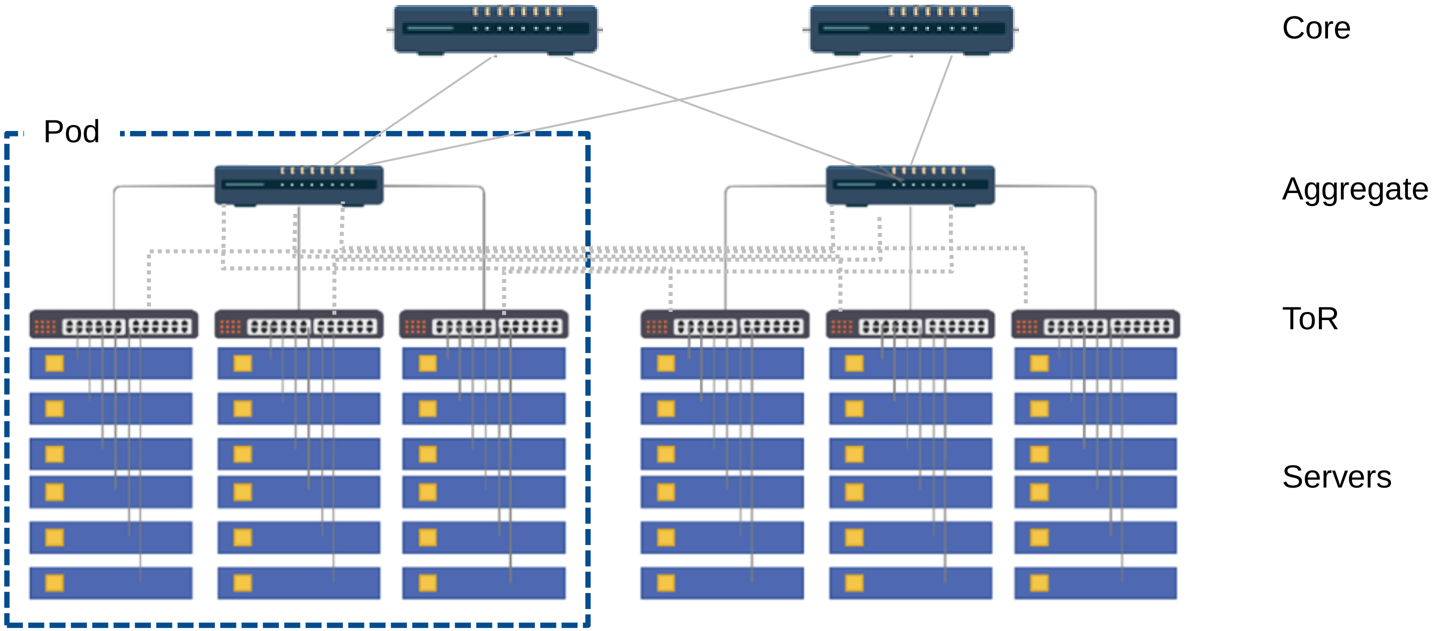



**How reliable is a system?**



# System reliability

- Aggregation/connected set of components
- First studies towards sovereignty→ data center use case
  - Best topology?
  - How many manufacturers?
  - How they should be placed?



# System reliability

## Data Centers (DC)

DC Modeling → Failure Generation → Sovereignty Analysis

## DC Topology

*3 Tier Leaf Spine*
*Fat Tree*
*AB-Fat tree*
*Facebook 4-post*
*Facebook Fabric*

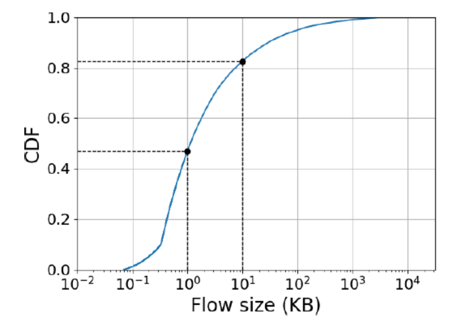
## Arrangement

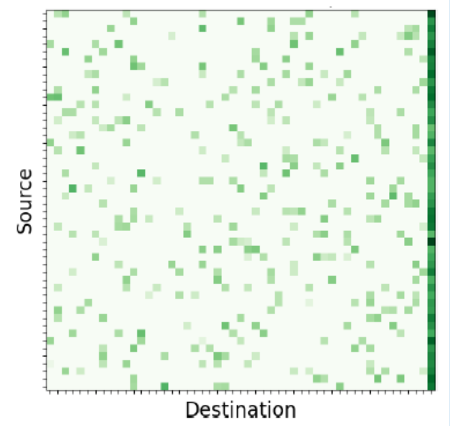
*Random*
*Left-Right*
*Left-Right Sequential*
*Pod-wise*

## DC Size

*Small (1K Servers)*
*Medium(32K Servers)*
*Large(64K Servers)*
*Mega(100K Servers)*

## Traffic



# System reliability

## Data Centers (DC)


Different failure scenarios:

- For each layer (ToR, aggregation, core)
- For each manufacturer/set of manufacturers
  - Hardware manufacturers
  - Software developers
    - Native developers
    - Other software developers

Evaluate the impact on the topology connectivity and survivable traffic.

# System reliability

## Data Centers (DC)


Heat maps and robustness surfaces on connectivity and max-flow between ToR pairs

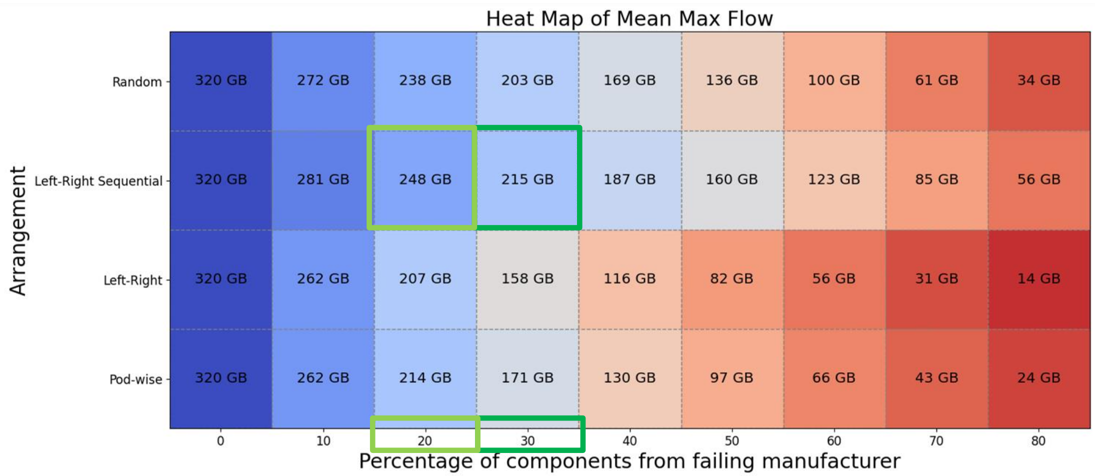
Compare

- Different topologies
- Different manufacturers
- Different arrangements

Evaluate sensitivity analysis

# System reliability

## Data Centers (DC)


Left-Right Sequential Best

If operator aims at survival traffic

210GB→ at least 3 manufacturers

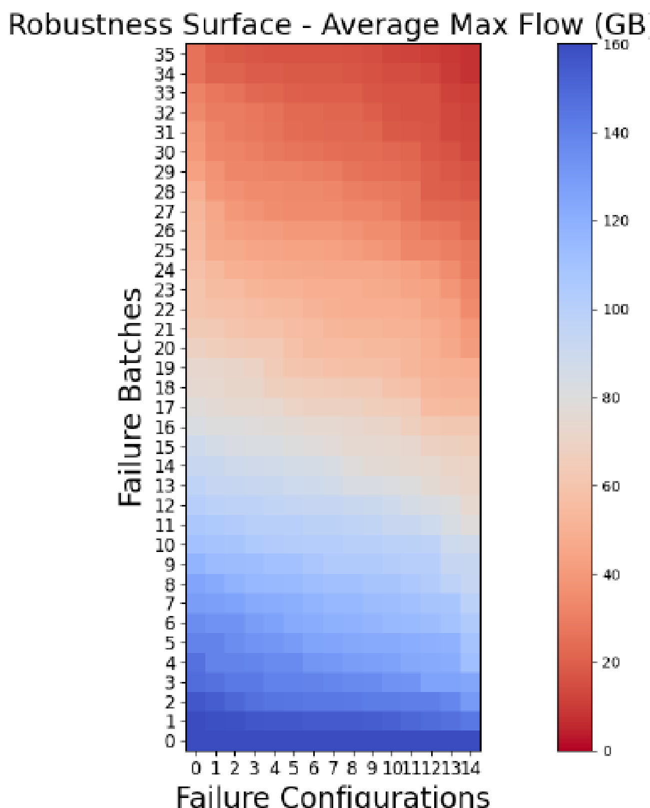
240GB→ at least 4 manufacturers

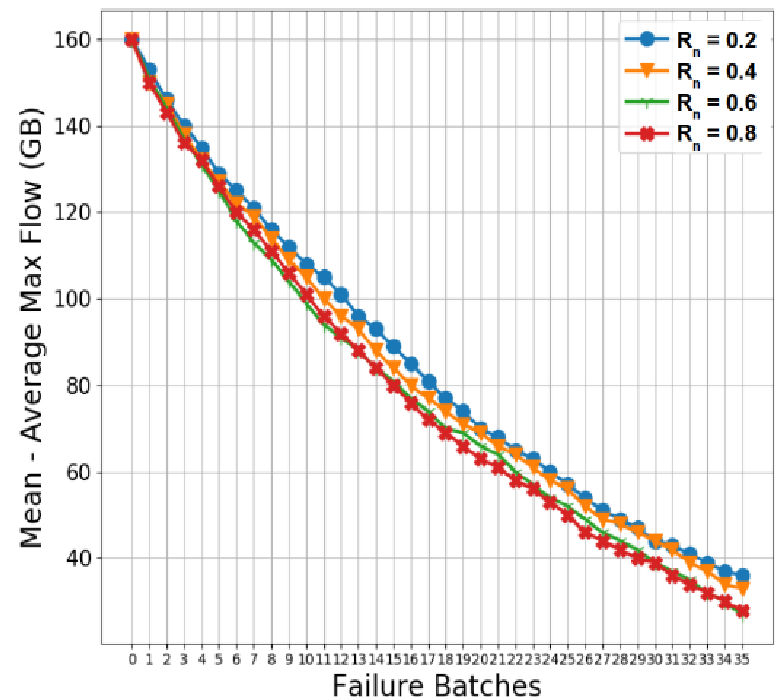
# System reliability

## Data Centers (DC)


5 HW manufacturers

5 SW manufacturers






*Rn: Ratio of man. SW dev. to all SW failures*

# Data center operators guidelines

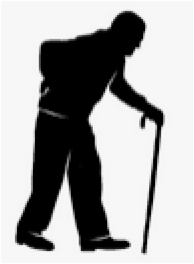
In small DCNs (less than 5000 servers)→ Leaf-Spine
In larger DCNs → Clos-network-based topology (e.g., fat tree)

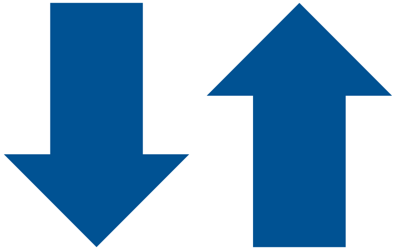
The higher the requirements, the more manufacturers are needed→ market and law limited

Severity of SW failures→ critical parameter to determine number of required developers

The more HW manufacturers, the less *non-native* SW developers are required

# Conclusions





Impact of software failures

Ageing and bugs

Presented bottom-up approach

Questions?